```
/*****************************************************************************
 Module
   Master.c

 Revision
   1.0.1

 Description
   This is the main state machine for implementing the C32 (Roamer Side) software.


*****************************************************************************/
/*---------------------------- Include Files ----------------------------*/
/* include header files for this state machine as well as any machines at the
   next lower level in the hierarchy that are sub-machines to this machine
*/
#include "HEADERS.h"

/*---------------------------- Module Defines ----------------------------*/


#define HeaderByte      0b01111110
#define DestinationMSB  0x20
#define DestinationLSB  0x80
#define TXOptions       0b00000000

#define RXFirst      9
#define RXSecond     10
#define RXRight      11
#define RXGripper    12
#define RXCamera        13
#define RXDigital    14

 /*---------------------------- Module Functions ----------------------------*/
static void SpecialCommInit(void );
static void SCI_Init(void );
static void InitTimers(void );
static void receiveSM(void );
static void sendSM(void );
static boolean sendByte(char);
static void processKeyboard(uint16_t input);

/*---------------------------- Module Variables ----------------------------*/
// with the introduction of Gen2, we need a module level Priority variable
static uint8_t MyPriority;

static unsigned short BatteryPeriod[MEMORY] = {0,0,0,0,0,0,0,0,0,0};// remember last 10
periods

static MasterState_t CurrentState;
static TXState_t CurrentTXState = TX_START;
// with the introduction of Gen2, we need a module level Priority var as well
static uint8_t MyPriority;
static char transmit[108];
static char receive[108];
static int currentRoamer;
static boolean receiving = FALSE;
static boolean sending = FALSE;
```

```
static boolean connect;
static boolean active;

static char Lamp1TX;
static char Lamp2TX;
static char Lamp3TX;
static char ChargeTX;
static char MyRoamerNumber;
static char LastCamCom;

static int ConnectedSourceMSB =0x00;
static int ConnectedSourceLSB =0x00;

static unsigned int LastFallingEdge;

static int connected = 0;   // 1 if connected
static int MessageType;

static int left, right;

static boolean cameraForward = TRUE;

/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
 Function
     InitTemplateService

 Parameters
     uint8_t : the priorty of this service

 Returns
     boolean, False if error in initialization, True otherwise

 Description
     Saves away the priority, and does any
     other required initialization for this service
 Notes

 Author
     J. Edward Carryer, 01/16/12, 10:00
****************************************************************************/
boolean InitMaster ( uint8_t Priority )
{
  ES_Event ThisEvent;
  MyPriority = Priority;
  /********************************************
   in here you write your initialization code
   ********************************************/
   // Call other intialization functions
   MotorDriveInit();
   SpecialCommInit();
   SCI_Init();
   InitTimers();

   DDRM = 0xFF-7;
   CHARGE_REG &= ~CHARGE_PIN;

   Servo12_Init(SERVOSTRING);//SERVOSTRING");
```

```
   // Analog Init
   ADS12_Init(ANALOGSTRING);     // using AD0 for command output , AD1 and AD2 are inputs
for which roamer

   MyRoamerNumber = 11; //Max- change for testing
  // post the initial transition event
   ThisEvent.EventType = ES_INIT;
   if (ES_PostToService( MyPriority, ThisEvent) == True)
   {
       return True;
   }else
   {
       return False;
   }
}

/****************************************************************************
 Function
     PostMaster

 Parameters
     EF_Event ThisEvent ,the event to post to the queue

 Returns
     boolean False if the Enqueue operation failed, True otherwise

 Description
     Posts an event to this state machine's queue
 Notes

 Author
     J. Edward Carryer, 10/23/11, 19:25
****************************************************************************/
boolean PostMaster( ES_Event ThisEvent )
{
  return ES_PostToService( MyPriority, ThisEvent);
}

/****************************************************************************
 Function
    RunMaster

 Parameters
   ES_Event : the event to process

 Returns
   ES_Event, ES_NO_EVENT if no error ES_ERROR otherwise

 Description
   add your description here
 Notes

 Author
   J. Edward Carryer, 01/15/12, 15:23
****************************************************************************/
ES_Event RunMaster( ES_Event ThisEvent )
{
```

```
  ES_Event ReturnEvent;
  static int u = 0;
  static int SpecialCommandState = 0;

  ReturnEvent.EventType = ES_NO_EVENT; // assume no errors


   if (ThisEvent.EventType == ES_NEW_KEY)
   {
      processKeyboard(ThisEvent.EventParam);

   }

  else if (ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam == SPECIAL_TIMER)
   {
     if(SpecialCommandState == 0)
     {
         SpecialCommandState = 1;
         COMMAND_PORT |= COMMAND_PIN;// set command pin high - update special comm about
Lamps
         // reset timer
           ES_Timer_SetTimer(SPECIAL_TIMER, 100);       // make sure we have enough time
to update
           ES_Timer_StartTimer(SPECIAL_TIMER);
     }

     else if(SpecialCommandState == 1)
     {
         SpecialCommandState = 0;

         COMMAND_PORT &= ~COMMAND_PIN;// set command pin high - update special comm about
Lamps
         // reset timer
           ES_Timer_SetTimer(SPECIAL_TIMER, 1000);       // make sure we have enough time
to update
           ES_Timer_StartTimer(SPECIAL_TIMER);
     }

      if(connected) PTAD |= BIT5HI;
      else PTAD &= BIT5LO;
   }



   else if (ThisEvent.EventType == ES_TIMEOUT && ThisEvent.EventParam == 3)
   {
      if (connected == 1)
      {
         ConnectedSourceMSB = 0x00;
              ConnectedSourceLSB = 0x00;
              SetMotorDriveDuty(128, MOTOR_RA);
              SetMotorDriveDuty(128, MOTOR_LA);
              connected = 0;
              active = FALSE;
              CurrentTXState = TX_START;
      }

   }
```

```
   else if (ThisEvent.EventType == ES_LAMP1)
   {
      Lamp1TX = ThisEvent.EventParam;

   }

   else if (ThisEvent.EventType == ES_LAMP2)
   {
      Lamp2TX = ThisEvent.EventParam;

   }

   else if (ThisEvent.EventType == ES_LAMP3)
   {
      Lamp3TX = ThisEvent.EventParam;

   }

   else if (ThisEvent.EventType == ES_CHARGE_STATE)
   {
      ChargeTX = ThisEvent.EventParam;

   }

  return ReturnEvent;
}

/****************************************************************************
 INIT FUNCTIONS
 ****************************************************************************/


static void SpecialCommInit()
{
   // output ports
   LAMP_OREG |= (LAMP1_OPIN |LAMP2_OPIN | LAMP3_OPIN);


   // input ports
   LAMP_IREG &= ~(LAMP1_IPIN | LAMP2_IPIN | LAMP3_IPIN);  // INPUTS
   CHARGE_REG &= ~CHARGE_PIN;


   //input capture setup
      TSCR1 = _S12_TEN; // turn timer subsystem on, doesn't stop timer in freeze
      TSCR2 = (_S12_PR2 |_S12_PR0);  // set to divde by 128
      TIOS &= ~_S12_IOS7; // set cap/comp 3 to input capture  (0 = input capture)
      TCTL3 |= (_S12_EDG7A |_S12_EDG7B); // capture rising and falling edges
      TFLG1 = _S12_C7F; // clear IC7 flag
      TIE |= _S12_C7I; // enable IC7 interrupt
     //EnableInterrupts(); // taken care of in Framework


}
```

```
static void SCI_Init()
{
   SCIBDH = 0;
   SCIBDL = 156; // set baud rate to 9600

   // control register 1
   SCICR2 = ( _S12_RIE | _S12_TE |_S12_RE); // transmit receive enable, transmit enable,
receive enable

   //SCICR2 |= _S12_TIE;// transmit interrrupt enable ON TIMEOUT

}

static void InitTimers()
{
   ES_Timer_SetTimer(SPECIAL_TIMER, 200);
   ES_Timer_StartTimer(SPECIAL_TIMER);

   //ES_Timer_SetTimer(1,500);
   //ES_Timer_StartTimer(1);

}


/*****************************************************************************
 COMM FUNCTIONS
 *****************************************************************************/

static boolean sendByte(char msg )
{
   if ((SCISR1 & _S12_TDRE) != 0)
   {
      SCIDRL = msg;
      return TRUE;
   } else
   {
      return FALSE;
   }

}

static void sendSM(void )
{
       static int checksum;
       static int FrameID = 0x01;
       static char byteToSend = 0x00;
       static double accx, accy, gx, gy;
       static int left, right;
       static int avg = 0;
       int test;
       int i;

       switch(CurrentTXState)
       {
          case TX_START:
                     if(sendByte(0x7E)) CurrentTXState = TX_MSBLength;
                     checksum = 0;
                     break;
```

```
                case TX_MSBLength:
                       if(sendByte(0x00)) CurrentTXState = TX_LSBLength;
                       break;
                case TX_LSBLength:
                       if(sendByte(0x0A)) CurrentTXState = TX_API;
                       break;
                case TX_API:
                       if(sendByte(0x01))
                       {
                          CurrentTXState = TX_FRAME;
                          checksum += 0x01;
                       }
                       break;
                case TX_FRAME:
                       if(sendByte(FrameID))
                       {
                          CurrentTXState = TX_MSBDestination;
                          checksum += FrameID;
                          if (FrameID > 50) FrameID = 0x01;
                       }
                       break;
                case TX_MSBDestination:
                       if(sendByte(ConnectedSourceMSB))
                       {
                          CurrentTXState = TX_LSBDestination;
                          checksum += ConnectedSourceMSB;
                       }
                       break;
                case TX_LSBDestination:
                       if(sendByte(ConnectedSourceLSB))
                       {
                          CurrentTXState = TX_OPTIONS;
                          checksum += ConnectedSourceLSB;
                       }
                       break;
                case TX_OPTIONS:
                       if(sendByte(TXOptions))
                       {
                          CurrentTXState = TX_STATUS;
                          checksum += TXOptions;
                       }
                       break;
                case TX_STATUS:
                   if(MessageType == DisconnectMessage && connected ==0)
                   {
                      ConnectedSourceMSB = 0x00;
               ConnectedSourceLSB = 0x00;
                       byteToSend = 0x05;
                   }
                   else if(MessageType == ConnectMessage && connected ==1) byteToSend =
0x04;
                   else byteToSend = 0x03;

                       if(sendByte(byteToSend))
                       {
                          CurrentTXState = TX_BATT;
                          checksum += byteToSend;
                       }
```

```
                    break;
          case TX_BATT:
          for(i = 0; i<10; i++)
          {
            avg = avg+((int)BatteryPeriod[i])/(MEMORY); // take average of last 10 battery
periods measured
          }
                avg = (avg/147*2);
                if (avg > 255) avg = 255;
          byteToSend = avg;
          avg = 0;
              if(sendByte(byteToSend))
              {
                 CurrentTXState = TX_CHARGE;
                 checksum += byteToSend;
              }
              break;
           case TX_CHARGE:
              byteToSend = ChargeTX;
              if(sendByte(byteToSend))
              {
                 CurrentTXState = TX_LAMP;
                 checksum += byteToSend;
              }
              break;

           case TX_LAMP:
          byteToSend = (Lamp1TX |Lamp2TX |Lamp3TX);
              if(sendByte(byteToSend))
              {
                 CurrentTXState = TX_CAMERA;
                 checksum += byteToSend;
              }
              break;
           case TX_CAMERA:
              byteToSend = LastCamCom;
              if(sendByte(byteToSend))
              {
                 CurrentTXState = TX_Checksum;
                 checksum += byteToSend;
              }
              break;
               case TX_Checksum:

                      if(sendByte(0xFF - (checksum & 0xFF)))
                      {
                         CurrentTXState = TX_START;
                         SCICR2 &= ~_S12_TIE;
                     sending = FALSE;

                      }
                      break;
               default:

           break;
        }
}
```

```
static void receiveSM(void )
{
   char input;

   static int i;
   static int checksumindex, checksum;
   static int MSBLength, LSBLength, Length, APIIdent, SourceMSB, SourceLSB, Source, RSSI,
Options;
   static int LEDBits, Status, Battery, Charge, Camera;
   static int correctsource = 0; // 1 if source is correct
   int left, right;
   char SpecialLedRX = 0;


   if ((SCISR1 & _S12_RDRF) != 0) // if interrupt enabled and ready to recieve
   {
      input = SCIDRL;
      input &= 0xFF;
   }

   if ((input & 0xFF) == HeaderByte && active == FALSE)
   {
      active = TRUE;

      i = 2;
      receiving = TRUE;
          receive[1] = HeaderByte;

   } else if (active == TRUE)
   {
      receive[i] = input; // i is the byte number
          switch(i)
          {
             case 2:

                        MSBLength = receive[i] & 0xFF;
                        break;
              case 3:
                     LSBLength = receive[i] & 0xFF;
                     Length = (int)(LSBLength);

                     if (Length < 1 || Length > 30)
                     {
                        Length = 3;
                     }
                     break;
              case 4:
                     APIIdent = receive[i] & 0xFF;
                     break;
              case 5:
                 correctsource = 1;
                 if(APIIdent == 0x81)
                 {
                    SourceMSB = receive[i] & 0xFF;
                    if (connected ==1)
                    {
```

```
                        if(SourceMSB != ConnectedSourceMSB)
                        {
                           correctsource =0;
                           active = FALSE;
                        }
                     }
                  } else if (APIIdent == 0x89)
                  {

                  }
                      break;
               case 6:

                      if(APIIdent == 0x81)
                  {

                     SourceLSB = receive[i] & 0xFF;
                         Source = (int)(SourceLSB + (SourceMSB << 8));

                         if (connected ==1)     // if connected to a POD, make sure this
is the correct POD
                     {
                        if(SourceLSB != ConnectedSourceLSB)
                        {
                           correctsource =0;
                           active = FALSE;
                        }
                     }

                  } else if (APIIdent == 0x89)
                  {

                  }
                      break;
               case 7:
                      if(APIIdent == 0x81)
                  {
                     RSSI = receive[i];
                  }

                      break;
               case 8:
                  if(APIIdent == 0x81)
                  {
                         Options = receive[i];
                  }
                      break;
               case RXFirst:

                      if (APIIdent == 0x81)  // figure out message type
                      {
                             if(receive[i] == 0x00)
                             {
                               MessageType = ConnectMessage;
                             }

                             else if(receive[i] == 0x01)
                             {
```

```
                                MessageType = DisconnectMessage;
                                if (connected == 0) correctsource = 0;
                                if (correctsource == 1 && connected == 1)
                                {

                                   connected = 0;
                                   SetMotorDriveDuty(128, MOTOR_RA);
                                   SetMotorDriveDuty(128, MOTOR_LA);
                                }
                             }

                             else if(receive[i] == 0x02)
                             {
                               MessageType = CommandMessage;
                               if (connected == 0) correctsource = 0;
                             }

                             if(correctsource == 0 && connected == 1) MessageType =
WrongSourceMessage;// we are getting a command
                          }
                          break;
                    case RXSecond:

                          if (APIIdent == 0x81)
                          {
                             left = (int)(receive[i] & 0xFF);
                             if(MessageType == CommandMessage && connected == 1 &&
correctsource == 1)
                             {
                                // left motor command
                                if(cameraForward)
                                {
                                   SetMotorDriveDuty(left, MOTOR_LA);
                                } else
                                {
                                   SetMotorDriveDuty(255-left, MOTOR_RA);
                                }


                             } else if(MessageType == ConnectMessage && connected == 0) // if
this is a connect message and we are not connected to another POD
                             {
                                if(receive[i] == 1)
                                {
                                   if((PTAD & BIT1HI) != 0)
                                   {
                                      ConnectedSourceLSB = SourceLSB & 0xFF;  // correct
roamer, save source address
                                   ConnectedSourceMSB = SourceMSB & 0xFF;  // Max- might also
want to move this after checking checksum?
                                   connected = 1;
                                   }
                                } else if (receive[i] == 2)
                                {
                                   if((PTAD & BIT2HI) != 0)
                                   {
                                      ConnectedSourceLSB = SourceLSB & 0xFF;  // correct
roamer, save source address
```

```
                              ConnectedSourceMSB = SourceMSB & 0xFF;  // Max- might also
want to move this after checking checksum?
                              connected = 1;
                              }
                           } else if (receive[i] == 3)
                           {
                              if((PTAD & BIT3HI) != 0)
                              {
                                 ConnectedSourceLSB = SourceLSB & 0xFF;  // correct
roamer, save source address
                              ConnectedSourceMSB = SourceMSB & 0xFF;  // Max- might also
want to move this after checking checksum?
                              connected = 1;
                              }
                           } else if (receive[i] == 11)
                           {

                              ConnectedSourceLSB = SourceLSB & 0xFF;  // correct roamer,
save source address
                           ConnectedSourceMSB = SourceMSB & 0xFF;  // Max- might also
want to move this after checking checksum?
                           connected = 1;

                           }
                           if (connected == 0)
                           {
                              correctsource = 0;
                           }
                        }

                     }
                     break;
               case RXRight:

                     if (APIIdent == 0x81)
                     {
                        right = (int)(receive[i] & 0xFF);
                        if(MessageType == CommandMessage && connected ==1 &&
correctsource ==1)
                        {
                           // right motor command

                           if(cameraForward)
                           {
                              SetMotorDriveDuty(right, MOTOR_RA);
                           } else
                           {
                              SetMotorDriveDuty(255-right, MOTOR_LA);
                           }

                        }
                     }
                     break;
               case RXGripper:
                     if (APIIdent == 0x81)
                     {
                           if(MessageType ==CommandMessage && connected ==1 &&
correctsource ==1)
```

```
                        {
                           // gripper command
                           Servo_SetAngle(GrabberServo,receive[i]) ;

                        }

                     }
                     break;
               case RXCamera:
                     if (APIIdent == 0x81)
                     {
                        if(MessageType ==CommandMessage && connected ==1 && correctsource
==1)
                        {
                           // gripper command
                           Servo_SetAngle(SpecialServo,receive[i] & 0xFF) ;
                           LastCamCom = receive[i];

                           if((LastCamCom & BIT7HI) != 0)
                           {
                              cameraForward = TRUE;
                           } else
                           {
                              cameraForward = FALSE;
                           }
                        }
                     }
                     //Set Servo angle here
                     break;

          case RXDigital:

                     if (APIIdent == 0x81)
                     {
                        //active = 0;
                            if(MessageType ==CommandMessage && connected ==1 &&
correctsource ==1)
                        {
                           SpecialLedRX = receive[i];

                     if((SpecialLedRX & BIT0HI) != 0) LAMP_OPORT |= LAMP1_OPIN; // led 1
on
                  else LAMP_OPORT &= ~(LAMP1_OPIN);

                  if((SpecialLedRX & BIT1HI) != 0) LAMP_OPORT |= LAMP2_OPIN; // led 2 on
                  else LAMP_OPORT &= ~(LAMP2_OPIN);

                  if((SpecialLedRX & BIT2HI) != 0) LAMP_OPORT |= LAMP3_OPIN; // led 3 on
                  else LAMP_OPORT &= ~(LAMP3_OPIN);

                        }
                     }

                     break;
               default:
                     break;
         }
                     //Process data if needed here
```

```
        if (i > Length + 3){ // Checksum

             checksumindex = 4;
             checksum = 0;


             for (checksumindex; checksumindex < i; checksumindex++){
                    checksum += receive[checksumindex];
                    if((checksum & 0xFF) == 0xFF)
                    {

                    }

          }
          if ((APIIdent == 0x81) && correctsource == 1)
          {
            sending = TRUE;
           SCICR2 |= _S12_TIE; // enable transmit

           ES_Timer_SetTimer(3,1000);
           ES_Timer_StartTimer(3);
          }


             active = FALSE;
             i = 1;

        }

        i++;
   }
}


 /*****************************************************************************
 intertupt functions
 *****************************************************************************/

void interrupt _Vec_TIMER7 BatLifeMeasure (void)
{

   char PinState;
   static unsigned int LastPinState;
   //int avg = 0;
   unsigned short CurrentTime;
   static unsigned int i =0;
   static unsigned short LastRisingEdge;
   unsigned short BatteryP;

   CurrentTime = TC7;
   TFLG1 = _S12_C7F; // clear IC7 Flag

   PinState = PTT & BIT7HI;  // read current pin state   -Max: changed to & instead of |

   if(PinState ==0) // it was a falling edge
   {
```

```
     BatteryPeriod[i] = (CurrentTime - LastRisingEdge); // encoder period is a module
level variable
     if(i <MEMORY-1) i++;
     else  i = 0;

   } else // if pin state is high
   {
   LastRisingEdge = CurrentTime;
   }

}


void interrupt _Vec_SCI SCI_Interrupt(void)
{

   if((SCISR1 & _S12_RDRF) != 0)
   {
      receiveSM();

   } else if ((SCISR1 & _S12_TDRE) != 0 && sending == TRUE)
   {
      sendSM();

   }

}
/*------------------------------ Footnotes -------------------------------*/
/*------------------------------ End of file -----------------------------*/
```